Embodied, On-line, On-board Evolution for Autonomous Robotics

A.E. Eiben, Evert Haasdijk, Nicolas Bredeche

Embodied, On-line, On-board Evolution for Autonomous Robotics

A.E. Eiben, Evert Haasdijk, Nicolas Bredèche

Artificial evolution plays an important role in several robotics projects. Most commonly, an evolutionary algorithm (EA) is used as a heuristic optimiser to solve some engineering problem, for instance an EA is used to find good robot controller. In these applications the human designers/experimenters orchestrate and manage the whole evolutionary problem solving process and incorporate the end result –that is, the (near-)optimal solution evolved by the EA– into the system as part of the deployment. During the operational period of the system the EA does not play any further role. In other words, the use of evolution is restricted to the pre-deployment stage.

Another, more challenging type of application of evolution is where it serves as the engine behind adaptation *during* (rather than before) the operational period, *without human intervention*. In this section we elaborate on possible evolutionary approaches to this kind of applications, position these on a general feature map and test some of these set-ups experimentally to assess their feasibility.

The main contributions of this section can be summarised as follows:

- It provides a taxonomy of evolutionary systems encountered in related work. This taxonomy helps to identify the essence of particular approaches, to distinguish them from each other, and to position various options for a robotics project.
- It offers the first results of experiments aiming at the practical evaluation of (some of) these options.

5.2.1 Controllers, Genomes, Learning, and Evolution

In this subsection we elaborate on the fundamental notions of controllers, phenotypes, genotypes, learning, and evolution. We do not aim for universally valid definitions of these concepts (if such things are possible at all), rather at a consistent taxonomy and terminology reducing the chances of mis-communication.

An essential design decision when evolving robot controllers is to distinguish phenotypes and genotypes regarding the controllers. Simply put, this distinction means that:

- We perceive the controllers with all their structural and procedural complexity as phenotypes.
- We introduce a (typically structurally simpler) representation of the controllers as genotypes.
- We define a mapping from genotypes to phenotypes, that might be a simple mapping, or a complex transformation through a so-called developmental engine.

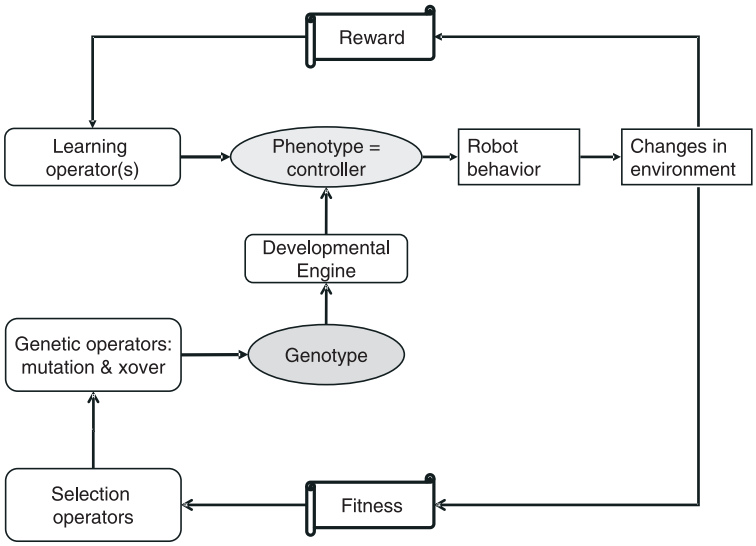

Fig. 5.9 General scheme of evolution and learning based on the genotype – phenotype distinction.

For example, a robot controller may consist of a group of artificial neural nets (ANNs) and a decision tree, where the decision tree specifies which ANN will be invoked to produce the robot's response in a given situation. This decision tree can be as simple as calling some ANN-1 when the robot is in stand-alone mode (not physically connected to other robots) and calling some ANN-2 when the robot is physically aggregated, i.e., connected to other robots. This complex controller, i.e., phenotype, could be represented by a simple genotype of two vectors, showing the weights of the hidden layer in ANN-1, respectively ANN-2. The two ovals and the link between them (including the developmental engine) in the middle of Fig. 5.9 shows this division.

A technical distinction between learning and evolution is now straightforward if we postulate that learning acts at phenotypic level, while evolution only affects the genotypes. As a consequence, we obtain two feedback-loops for adaptation, a learning loop and an evolutionary loop as shown in Fig. 5.9. There are a couple of things to be noted about this scheme. First, note that, just like the learning loop, the evolutionary feedback-loop includes the controllers, since the genotypes do not interact directly with the environment. Instead, the genotype determines the phenotype (controller) which in turn determines the robot behaviour. This behaviour in turn causes changes in the environment, including other robots. Second, both loops involve a utility measure, required to direct adaptation. For reasons of clarity we distinguish these measures also by name, using the term reward for learning and the term fitness for evolution. Third, please note that this simple diagram might become more complicated through the addition of more complex interactions. For instance, using

Lamarckian operators we obtain learning on the genotype level, while an additional social learning mechanism can be naturally perceived as an evolutionary process on the phenotype level. Having noted all this, and keeping possible variations in mind, in essence we distinguish learning and evolution by their point of impact and the related time scale. Simply put, learning acts on phenotype level on the short term, within the lifetime of an individual (robot controller), while evolution works on genotypes on the long term, over consecutive generations.

5.2.2 Classification of Approaches to Evolving Robot Controllers

There are a number of features that allow us to position evolutionary robotics approaches. In (Schut *et al*., 2009) we elaborated on the notion of situated evolution so that we could clarify similarities and differences between, for instance, regular GAs, spatially structured GAs, evolving ALife systems and evolutionary robotics. We now zoom in on evolutionary robotics and discuss different approaches by specifying a set of descriptive features and identifying a particular approach by the combination of features it belongs to. A previous attempt in this direction by Watson *et al*. offers a classification scheme in (Watson *et al*., 2002). This scheme is exhibited in Fig. 5.10. The figure shows that the primary distinction, i.e., the topmost junction in the graph, is based on embodiedness. This is also a key notion for us, since we are to apply evolution in real physical robots, hence in an embodied fashion. However, we want to go further in embodying evolution than doing the fitness evaluations through "embodied trials". We also want to embed the management and execution of evolutionary operators for selection and variation (i.e., mutation and crossover) in the robots. For a precise terminology we need to distinguish two essential components of an evolutionary process: the fitness evaluations, a.k.a. trials, on the one hand and the evolutionary operators on the other hand. Then we can also distinguish two basic types of embodied evolution: one where the fitness evaluations are embodied and one where the (management and execution of) evolutionary operators are embodied. The most common interpretation of the term embodied evolution coincides with the first case. Therefore, to prevent confusion, we will avoid using this term for the second case and will use the term on-board/intrinsic as introduced below.

Our classification scheme is based on a set of three features concerning the evolution of controllers from temporal, spatial, and procedural perspective. That is, we distinguish types of evolution considering when it happens, where it happens, and how it happens:

1. off-line or design time vs. on-line or run time (when),
2. on-board or intrinsic vs. off-board or extrinsic (where),
3. in an encapsulated or centralised vs. distributed manner (how).

Note, that we do not include embodiedness (of fitness evaluations) in this classification scheme. The reason is that the system we have in mind is one with real robots, where fitness evaluation always happens in reality. In other words, our whole

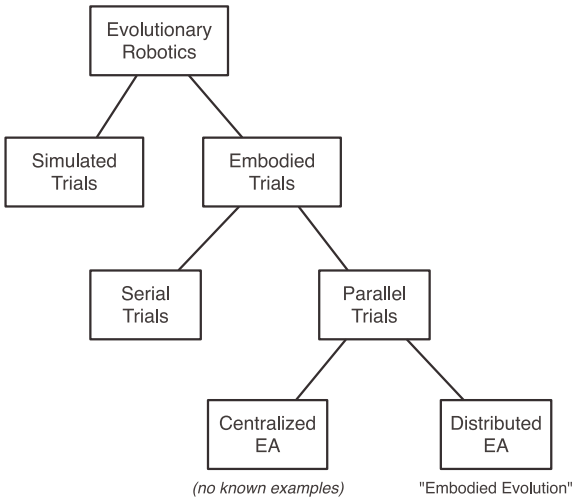

Fig. 5.10 Classification of evolutionary robotics approaches from (Watson *et al*., 2002).

scheme falls in the category of embodied evolution in the terminology after Watson *et al*.

In *off-line* evolution, the evolutionary development[13] of robot controllers takes place *before* the robots start their "real" operation period. *On-line* evolution is the opposite in that the evolutionary development of robot controllers takes place *during* the "real" operation period of the robots (although off-line evolution might precede on-line evolution as an educated initialisation procedure) and is an ever-continuing process. Obviously, the distinction between these two options lies in the release moment when the evolved controllers are deployed in the robots. If the evolutionary operators are no longer applied after the release moment and the controllers remain fixed (or only change by other mechanisms), we are dealing with the off-line case, otherwise we have on-line evolution.

From the spatial perspective, we distinguish the *on-board* or *intrinsic* case where the evolutionary operators such as selection, crossover, and mutation are performed exclusively inside the actual robot hardware, from the *off-board* or *extrinsic* case, where they are performed with the help of external equipment outside the robots. Such external equipment could be a computer, interfaced with the robots, that plays the role of "puppet master" in an on-line evolutionary process: based on fitness information it collects from the robots (embodied trials!) it manages the evolutionary operators for selection and variation and injects newly produced controllers into the existing robot bodies. If we view a system such as this in terms of parallel EAs –where there are many corresponding considerations– we would describe it as a master-slave parallel EA with the slaves calculating fitness and the master orchestrating evolution. Here we can recall our elaboration on embodiedness in the

[13] Development is meant here in the engineering sense, the "making of" or "building of" controllers, rather than the biological, embryo-genetic sense as "creating it from an embryo". We use the term development to hint at the iterative nature of this process.

beginning of this section. As we observed there, it would be formally correct to describe what we call on-board or intrinsic evolution as embodied evolution because the evolutionary operators are embodied in the robots. The reason to choose other terms here is twofold. First, the usual terminology associates embodied evolution with embodied trials, which is a completely different thing. Second, introducing new terms here facilitates precise phrasing: embodied evolution means that fitness evaluations (trials) are done in real-life by the robots, while on-board or intrinsic evolution means that the evolutionary operators executed by the robots.

Last, but not least, we consider how the evolutionary operators are managed. First, we distinguish the *distributed* approach, where each robot carries one genotype and is controlled by the corresponding phenotype. Robots can reproduce autonomously and asynchronously to create offspring controllers by recombination and/or mutation. Here, the iterative improvements (optimisation) of controllers result from the evolutionary process that emerges from the interactions between the robots. In terms of parallel EAs, such a distributed system is analogous to a cellular parallel EA. This approach is complemented by the *encapsulated* or *centralised* approach: each robot has a completely autonomous EA implemented on-board, maintaining a population of genotypes inside itself. These EAs can be different for different robots and are executed in a traditional, centralised manner locally, inside each robot. This is typically done by a time-sharing system, where one genotype from the inner population is activated (i.e., decoded into a phenotype controller) at a time and is used for a while to gather feedback on its quality. Here, the iterative improvements (optimisation) of controllers are the result of the EAs running in parallel inside the individual robots independently. In terms of parallel EAs, such a distributed system is analogous to an island-model parallel EA (without migration). Observe, that both the encapsulated and the distributed approaches yield a population of heterogeneous robot controllers. Furthermore, it is important to note that we distinguish distributed and centralised control of the EA, not of not the robots per se: they perform their tasks autonomously in all cases. Finally, a remark on the term centralised and encapsulated as defined here. To some extent, we use them as synonyms, both being the counterpart of the distributed approach. Strictly speaking the adjective "centralised" would already suffice, but we also introduce "encapsulated" to emphasise the fact that a (centralised) evolutionary algorithm is running entirely inside a robot.

Fig. 5.11 shows a classification graph along the lines described here. At first glance, this might seem at odds with the one in Fig. 5.10. However, the distinction between on- and off-line renders the two dichotomies based on "trials" (simulated vs embodied and serial vs parallel) superfluous: on-line adaptation only makes sense in real robots (although for experimental purposes, the whole system may be simulated): adaptation takes place as the robots go about their tasks and performance evaluation is inherently parallel across robots. Thus, the distinction between extrinsic and intrinsic EA operators matches the one between centralised and distributed EA and the further distinctions under the intrinsic case could be seen as a refinement of the "Distributed EA" leaf in Fig. 5.10.

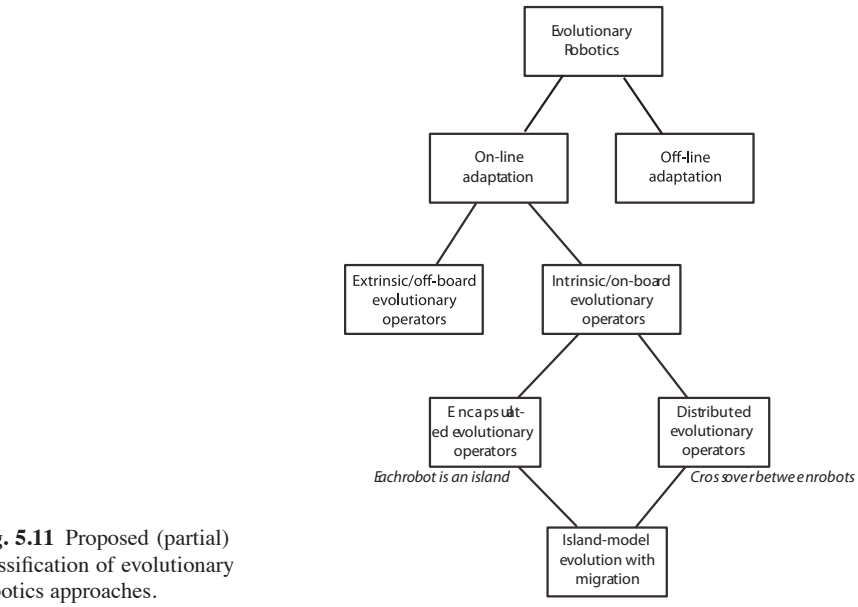

Fig. 5.11 Proposed (partial) classification of evolutionary robotics approaches.

5.2.3 The Classical Off-line Approach Based on a Master EA

Using the classification scheme based on the three features we have discussed it is possible to characterise existing approaches to evolutionary robotics (Floreano *et al*., 2008a; Nolfi & Floreano, 2000a). The usual approach to ER is to use a conventional EA for finding good controllers in a fashion that can be identified as

- off-line,
- extrinsic (off-board)
- centralised (encapsulated in an external computer).

Fig. 5.12 illustrates this approach. Note, that the arrow from the external computer to the robot represents the final deployment of the best found controller after the evolutionary search is finished. The figure does not show how the fitness evaluations are done during the evolutionary search. In other words, this figure covers the possibilities of evaluations in simulation as well as in real-life, i.e., in an embodied fashion.

The on-line evolutionary system we have in mind is radically different from this approach in that adaptation of the robots never stops. From our perspective, this means that evolution is being performed on-the-fly, possibly combined with other adaptive processes, such as individual learning or social learning. In the next section we discuss a number possible systems for on-line evolution.

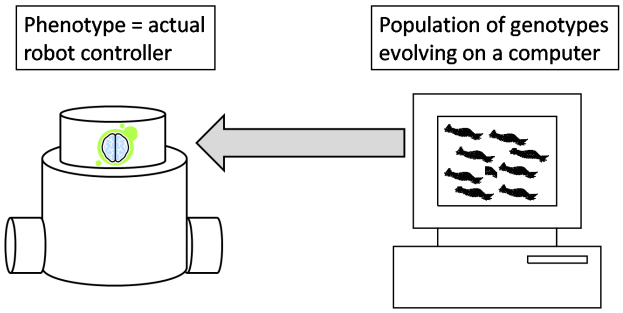

Fig. 5.12 The common off-line approach to evolutionary robotics, where the population of genotypes is evolved on a computer that executes the evolutionary operators (variation and selection), managed in a centralised fashion. As for fitness evaluation, the computer can invoke a simulator or send the genotype to be evaluated to a robot to test it (embodied evaluation). Evaluation of genotypes can happen in parallel. At the end of the evolutionary process the best genotype is deployed in the robot(s).

5.2.4 On-line Approaches

5.2.4.1 Encapsulating the EA in the Robots

This option amounts to implementing the EA to the robots to run it inside the robots while they are operating. Obviously, this implies a whole population of genotypes being hosted in one robot, while the robot can have only one controller at a time. This means that at any given time only one of the genotypes is activated, i.e., decoded into a controller. Fig. 5.13 illustrates this matter.

We will use the term encapsulated EA to designate this approach[14]. This approach is seldomly used with only two examples we know of (Nehmzow, 2002; Usui & Arita, 2003). Such a system, illustrated in Fig. 5.14, can be described as

- on-line,
- on-board (intrinsic),
- encapsulated (centralised).

The most natural option, matching the on-line character of this set-up, is in vivo fitness evaluation of genotypes by transforming them into phenotypes and using them to control the given robot for a while. After this evaluation period, another genotype can be transformed into phenotype/controller to undergo its own evaluation. Thus, all robots run their own EA on-the-fly, so we have a number of parallel evolutionary processes running independently as shown in Fig. 5.14.

[14] In (Nehmzow, 2002) this is called “embedded”, but we feel that embodied and embedded can be easily confused, therefore choose “encapsulated”.

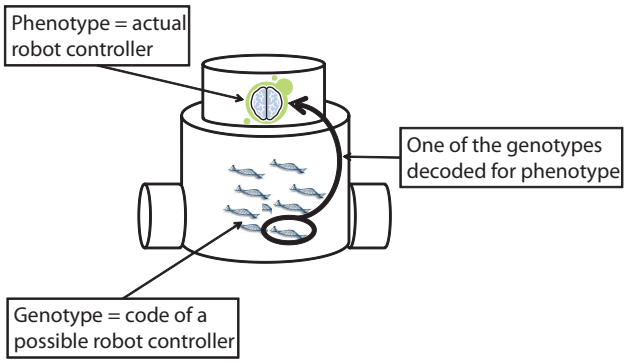

Fig. 5.13 Encapsulated evolution illustrated in one robot hosting an evolving population of genotypes. At any given time one of these genotypes is activated, i.e., decoded into a controller. Execution of evolutionary operators (variation and selection) takes places inside the robot, on-board, managed in a centralised fashion. Fitness evaluations are typically performed by activating the genotypes one by one through a time-sharing system and using them for a while.

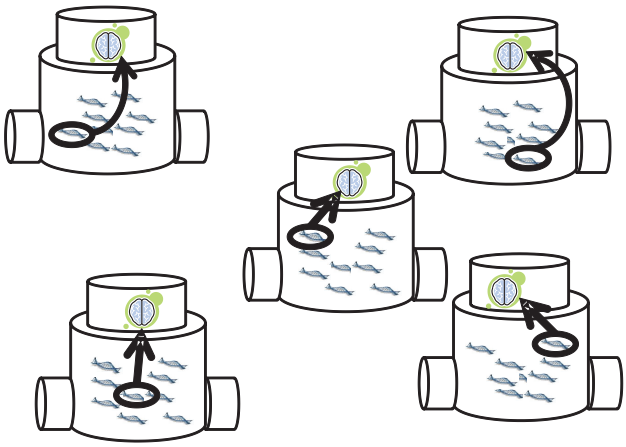

Fig. 5.14 Encapsulated evolution in a group of robots, where each robot is running a (centralised) evolutionary algorithm on-board independently. The evolutionary process does not require communication and interaction between robots.

5.2.4.2 Distributed Evolution

Our next example illustrates a set-up that is similar to ALife-like evolution with natural selection and natural reproduction (Eiben *et al.*, 2007). The difference with such ALife systems is caused by the practical constraint that robot bodies do not multiply. This implies that we have a fixed number of placeholders for controllers, the bodies, hence we cannot add a new controller to the population without removing an old one. Death of a controller without immediate replacement is in principle

possible, but would amount to a waste of resources (inactive robot), thus we expect a mechanism to prevent this. This all means that we have a "half-natural" reproduction, where reproduction and survivor selection are not independent, but mating is autonomous and asynchronous. Using our feature set this approach can be described as

- on-line,
- on-board (intrinsic),
- distributed.

Obviously, a decentralised system lacks a global puppet master orchestrating the process of evolution. Rather, the evolutionary process is a result of activities of the individual robots. In other words, all evolutionary operators for selection and reproduction are managed autonomously. Fig. 5.15 illustrates this type of on-line evolution.

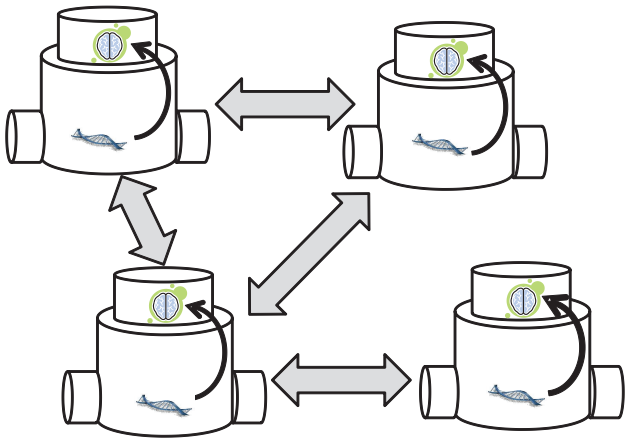

Fig. 5.15 Distributed on-board evolution, where each robot carries its own single genotype and evolution emerges from the reproductive interactions between robots. These interactions are indicated by the grey arrows.

5.2.4.3 Distributed and Encapsulated Evolution

Obviously, it is possible to combine the mechanisms of encapsulated and distributed evolution. From the encapsulated EA perspective, this means extending the system with migration of genotypes between robots. In this case we obtain interacting evolutionary processes, similar to the island model with migration for parallel evolutionary algorithms. This option is depicted in Fig. 5.16. Using our feature set this approach can be described as

- on-line,
- on-board (intrinsic),
- distributed *and* encapsulated.

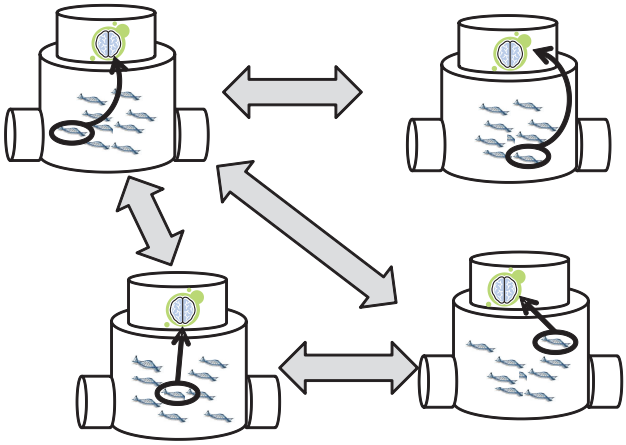

Fig. 5.16 On-board evolution where each robot is running an evolutionary algorithm inside and genotypes can migrate between robots. Execution of evolutionary operators takes places inside the individual robots, but communication and interaction between robots is required for the migration of genotypes. These interactions are indicated by the grey arrows.

5.2.4.4 Master EA Orchestrating On-line Evolution

The basis of this approach is the existence of a central authority to manage the evolutionary operators for selection and reproduction, while running in the on-line mode. Technically, this means that the given group of robots acts as a group of slaves (purely in terms of the EA). In this (heterogeneous) group each robot carries one genotype and sends fitness information to the master. Then it is the master who decides –using the global information it possesses– which robot controllers are to be recombined and/or mutated and which ones should be replaced with newly created controllers. The creation of new genotypes can take place inside this computer and the result deployed in the robots whose controller is selected for replacement. The genotype sent by the master is decoded/activated into a phenotype, i.e., into a working controller. From the perspective of the master this means that fitness evaluations can be done in real life and in parallel. If the group size of the robots equals the population size within the EA then the whole population can be evaluated simultaneously. From the perspective of the robots this means that they form a heterogeneous group and their controllers are repeatedly replaced by new ones – that might be better or worse than the one used before.

In terms of the classification scheme of Watson *et al*. in Fig. 5.10, this option belongs to the leaf on the path Embodied Trials – Parallel Trials – Centralised EA, with no known examples. The describing properties of this system are:

- on-line,
- extrinsic (off-board),
- centralised (encapsulated in an external computer, not in the robots).

A possible argument for using such a system is that the global information of the master and its ability to fully control selection and reproduction makes it easier to evolve good controllers than using a decentralised architecture. This type of evolution is illustrated in Fig. 5.17.

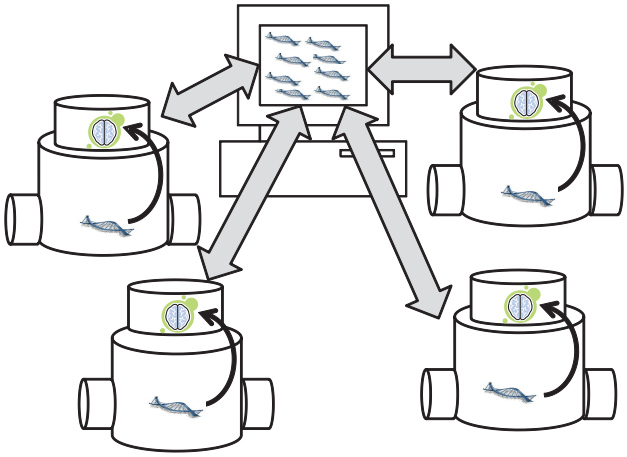

Fig. 5.17 Off-board on-line evolution where each robot carries one genotype and evaluates its fitness by using the controller it encodes. Evolution is managed by an external master computer that executes all evolutionary operators for selection and reproduction. Communication between the robots and the master is required for transmitting genotypes and fitness information. These interactions are indicated by the grey arrows in the figure.

5.2.5 Testing Encapsulated Evolutionary Approaches

In this section we report on the first experiments with encapsulated evolution.

5.2.5.1 The $(\mu + 1)$-ONLINE Evolutionary Algorithm

These experiments validate an encapsulated EA that is based on the classical $(\mu + 1)$ evolution strategy (Schwefel, 1995), where a population of μ individuals is main-

tained within each robot.[15] Here, each individual is the genotypic code of a robot controller that is in the form of an artificial neural net (ANN). This ANN is a perceptron with a hyperbolic tangent activation function using 9 input nodes (8 sensor inputs and a bias node), no hidden nodes and 2 output nodes (the left and right motor values), 18 weights in total. These 18 weights are to be evolved, therefore, the evolutionary algorithm will use the obvious representation of real-valued vectors of length 18 for the genomes. For the first experiments we decide to set the population size $\mu = 1$, restricting ourselves to a $(1 + 1)$ evolution strategy and consequently we omit recombination. We use a straightforward Gaussian mutation, adding values from a distribution $\mathsf{N}\ (0, \sigma)$ to each x_i in the genotype $\bar{x}$. This simple scheme defines the core of our EA, but it is not sufficient to cope with a number of issues in our particular application. Therefore, we extend this basic scheme with a number of advanced features, described below.

1. **Adapting σ values.** A singleton population is inherently very sensitive to premature convergence to a local optimum. To overcome this problem, we augment our EA with a mechanism that varies the mutation step-size σ on the fly, switching back and forth between local and global search, depending on the course of the search. In particular, σ is set to a pre-defined minimum to promote local search whenever a new genome is created. Then, σ gradually increases up to a maximum value (i.e., the search shifts towards global search) as long as the no improvements are found to the best genome found so far (the so-called champion, stored in the robot's archive). If local search leads to improvements, σ remains low, thus favouring local search. Otherwise the increasing σ values will move the search into new regions in the search space.
2. **Recovery period.** Because we use *in vivo* fitness evaluation, a new genome needs to be "activated" to be evaluated: it has to be decoded into a controller and take control of the robot for a while. One of the essential design decisions is to avoid any human intervention during evolution, such as repositioning the robot before evaluating a new genome. Consequently, a new controller will start where the previous one finished, implying the danger of being penalised for bad behaviour of its predecessor that may have manoeuvred itself into an impossibly tight corner. To cope with this effect, we introduce a *recoveryTime*, during which robot behaviour is not taken into account for the fitness value computation. This favours genomes that are efficient at both getting out of trouble during the recovery phase and displaying efficient behaviour during the evaluation phase.
3. **Re-evaluation.** The evaluation of a genome is very noisy because the initial conditions for the genomes vary considerably: an evaluation must start at the final location of the previous evaluation, leading to very dissimilar evaluation conditions from one genome to another. For any given genome this implies that the measurement of its fitness, during the evaluation period, may be misleading, simply because of the lucky/unlucky starting conditions. To cope with such noise, we re-evaluate the champion (i.e., current best) genome with a probability $P_{re-eavulate}$.

[15] Note that the population size μ within the EA should not be confused with the group size, i.e., the number of robots in the arena.

This is, in effect, a resampling strategy as advocated by Beyer to deal with noisy fitness evaluations (Beyer, 2000). As a consequence, the robots needs to share its time between producing and evaluating new genomes and re-evaluating old ones. The fitness value that results from this re-evaluation could be used to refine a calculation of the average fitness of the given genome. However, we choose to overwrite the previous value instead. This may seem counterintuitive, but we argue that this works as a bias towards genomes with low variance in their performance. This makes sense as we prefer controllers with robust behaviour. It does, however, entail an intrinsic drawback as good genomes may be replaced by inferior, but lucky genomes in favourable but specific conditions. Then again, a lucky genome which is not good on average will not survive re-evaluation, avoiding the adaptive process getting stuck with a bad genome.

The resulting method is called the $(\mu + 1)$-ONLINE evolutionary algorithm; its pseudo code is shown in Algorithm 5.

5.2.5.2 $(1 + 1)$ Encapsulated Evolution in a Hybrid Set-up

These results have been published more extensively in (Bredeche *et al*., 2009), therefore here we only give a brief summary of the method and the most important outcomes.

In the first series of experiments we tested the $(\mu + 1)$-ONLINE algorithm in a hybrid set-up that features actual robotic hardware, a Cortex M3 board with 256kb memory. This controls a simulated autonomous e-puck in a Player/Stage environment. After N time-steps, the evaluation of the current controller is complete and the controller parameters are replaced with values from a new genome, which is evaluated *from the location the previous controller left it in*. This means that *no* human intervention is *ever* needed. We run the experiment 12 times.

Fig. 5.18 illustrates the experimental set-up, with a Cortex board connected to the computer running Player/Stage. The simulated robot is modelled after an ePuck mobile robot with two wheels and eight proximity sensors. The maze environment used in our experiment is as shown in this figure.

For each run of the experiment, the robot starts with a random genome and a random seed. The fitness function promotes exploration and is inspired by a classic one, described in (Nolfi & Floreano, 2000a) which favours robots that are fast and go straight-ahead, which is of course in contradiction with a constrained environment, implying a trade-off between translational speed and obstacle avoidance. Equation 5.1 describes the fitness calculation:

$$fitness = \sum_{t=0}^{evalTime} (speed_{translational} \cdot (1 - speed_{rotational}) \cdot (1 - minSensorValue)) \tag{5.1}$$

The overview of experimental details is given in Table 5.2.

To provide an indication of the true performance and reusability of the best individuals found by $(\mu + 1)$-ONLINE evolution, a hall-of-fame is computed during

Algorithm 5: The $(\mu + 1)$-ONLINE evolutionary algorithm.

```
   // Initialisation
 1 for i = 1 to μ do
 2     population[i] = CreateRandomGenome
 3     population[i].Fitness = Fitness_min
 4 end
 5 for evaluation = 1 to N do
 6     Parent = SelectRandom(population)
 7     if random() < P_re-evaluate then
           // Don't create offspring, but re-evaluate selected
              parent itself
           // Get out of bad situations due to previous evaluation
 8         Recover(Parent)
           // Combination depends on re-evaluation strategy:
           // overwrite, average or exponential moving avg.
 9         Parent.Fitness = Combine(Parent.Fitness, RunAndEvaluate(Parent))
10         Sort(population)
11     end
12     else
           // Create offspring and evaluate that as challenger
13         Challenger = Mutate(Parent, Parent.σ)
           // Get out of bad situations due to previous evaluation
14         Recover(Challenger)
15         Challenger.Fitness = RunAndEvaluate(Challenger)
16         if Challenger.Fitness > population[μ].Fitness then
17             population[μ] = Challenger
18             population[μ].Fitness = Challenger.Fitness
19             population[μ].σ = σ_min
20             Sort(population)
21         end
22         else
23             Parent.σ = Parent.σ · 2
24         end
25     end
26 end
```

the course of evolution from the champions of all runs. The 10 best genomes from the hall-of-fame are validated by running each from six initial positions in the environment, indicated in Fig. 5.18. Starting from each of these positions, the genomes are evaluated for ten times the number of steps used for evaluation during evolution. Note, that one of the validation starting positions has certainly never been visited during development (test no.4, within a small enclosed area) and provides an extreme test case in a very constrained environment. This decomposition into an evolution (development) phase and a post-experiment testing phase is similar to the learning and testing phases commonly seen in Machine Learning and does not imply a deployment phase as in traditional, off-line evolutionary robotics approaches.

We conducted a series of twelve independent experiments $(\mu + 1)$-ONLINE evolution, with parameters set as stated above. Each experiment started with a different

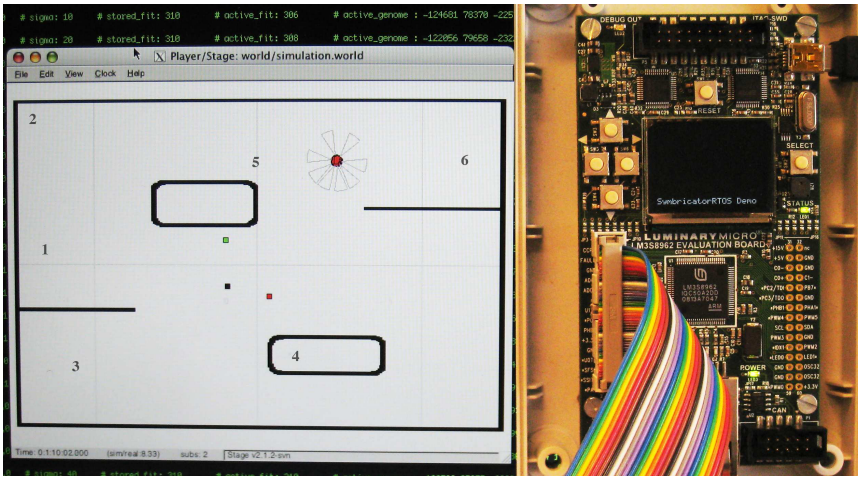

Fig. 5.18 The experimental setup: the Cortex board connected to Player/Stage. The numbers in the player-stage arena indicate the starting positions for the validation trials.

random controller (with very poor behaviour indeed) and a different random seed. The experiments ran for 500 evaluations and displayed different overall fitness dynamics with very similar patterns. In all our experiments, we saw a similar pattern of initial random search characterised by many different genomes with poor fitness; then, local search characterised by subsequent genomes with increasing fitness until a robust genome is found that survives re-evaluation for some time and then a switch to another region that yields good results or towards an inferior genome that got lucky (almost a restart, in effect).

During the course of the experiments a hall-of-fame was maintained for further validation of the best genomes. Fig. 5.19 shows the results of the validation of the hall-of-fame for three different schemes for re-evaluation: overwrite-last-fitness, where the champion's fitness is overwritten after every re-evaluation, average-fitness, where the fitness is the average of all re-evaluations and a scheme where there is no re-evaluation at all. This allows us to assess two things: whether high ranking genomes in the hall-of-fame are also efficient in a new set-up and whether the "overwrite fitness" re-evaluation scheme is relevant. The y-axis shows the normalised performance: the best of all individuals for a scenario is set to 1.0, the performance of the other individuals is scaled accordingly. For each scenario (arranged along the x-axis), the graphs show a mark for each individual from the hall-of-fame. All results for a given genotype are linked together with a line.

The graphs clearly show that re-evaluation improves performance substantially; from the ten best solutions without re-evaluation, only a single one performs at a level comparable to that of the ones with re-evaluation. It is harder to distinguish between the two algorithm variants using re-evaluation: averaging the fitness measurements for the genome in question or overwriting the archived fitness value with

Table 5.2 Experiment description table for the (1+ 1)-ONLINE evolution tests

Experiment details	
Task	fast forward
Arena	see Fig 5.18
Robot group size	1
Simulation length	1000 time steps
Controller details	
ANN type	perceptron
Input nodes	9 (8 sensory inputs and 1 bias node)
Output nodes	2 (left and right motor values)
Evolution details	
Representation	real valued vectors with $-4 \leq x_i \leq 4$
Chromosome length L	18
Fitness	See equation 5.1
Recovery time	30 time steps
Evaluation time	30 time steps
$P_{reevaluate}$	0.2
Population size μ	1
Mutation	Gaussian $N(0, \sigma)$ with adaptive σ values, $\sigma_{initial} = 1$
Crossover	n/a
Parent selection	n/a
Survivor selection	replace when better

the last measurement. On the one hand, the spread of performance *seems* greater for the case with averaging fitness than it does for overwriting fitness, which would endorse the reasoning that overwriting after re-evaluation promotes individuals with high average fitness and low standard deviation. On the other hand, however, the nature of real world experiments have a negative impact on the amount of data available for statistically sound comparison of re-evaluation strategies, as is often the case with real hardware, and keep from formulating a statistically sound comparison.

Further analysis of the ten best individuals with the overwrite-fitness re-evaluation scheme shows that the controllers actually display different kinds of behaviour – all good, robust, but different wall avoidance and/or open environment exploration strategies, ranging from cautious long turns (reducing the probability of encountering walls) to exploratory straight lines (improved fitness but more walls to deal with). Fig. 5.20 illustrates this by showing the pathways of these individuals, starting from an initial position on the left of the environment. This reflects the genotypic diversity observed in the hall-of-fame and hints at the algorithm's capability to produce very different strategies with similar fitness.

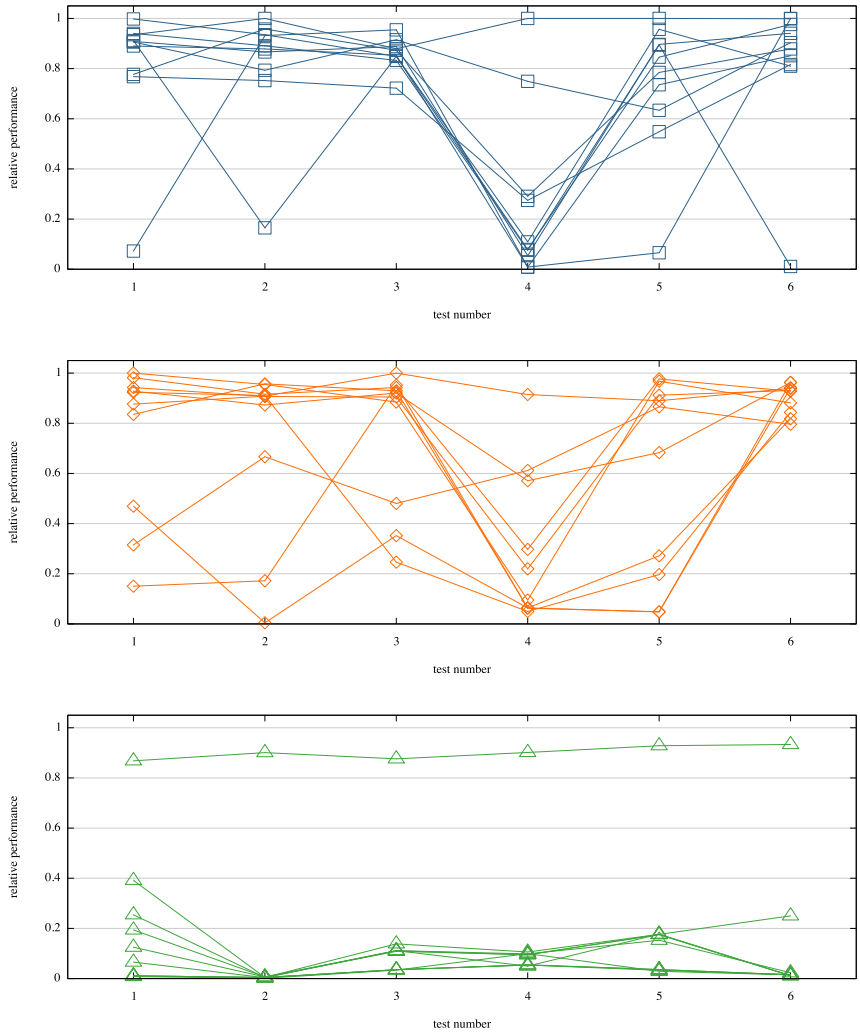

Fig. 5.19 Performance on validation scenarios for various re-evaluation schemes. Top: overwrite-last-fitness scheme; Middle: average-fitness scheme; Down: no re-evaluation scheme. X-axis shows the results on the six different validation setup (see Fig. 5.18), y-axis shows normalised fitness performance for each run. For a given genome, results in the six validation set-ups are joined together with a line.

5.2.5.3 $(\mu + 1)$-ONLINE Evolution in a Simulated Set-up

The second series of experiments increases the population size μ beyond 1. These experiments are performed in a pure simulation environment (Delta3D-based) util-

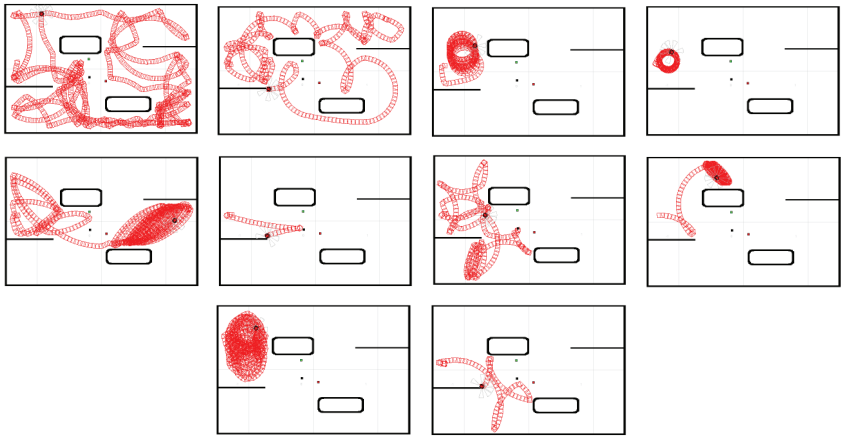

Fig. 5.20 Traces for the ten best controllers (using fitness replacement after re-evaluation).

ising four different arenas shown in Fig. 5.21. An additional difference with the first series of experiments is that here we use a group of five robots that are active in the given arena simultaneously. The robots obviously pose additional, moving obstacles for each other, but do not otherwise interact. For each arena and value of μ, we conducted 10 trials. To keep the experimental setup as close as possible to the previous one, we decided not to use crossover and not to use fitness-based parent selection either, because none of them was possible in the $(1+1)$ case. In this way we can study the effect of population size in isolation.

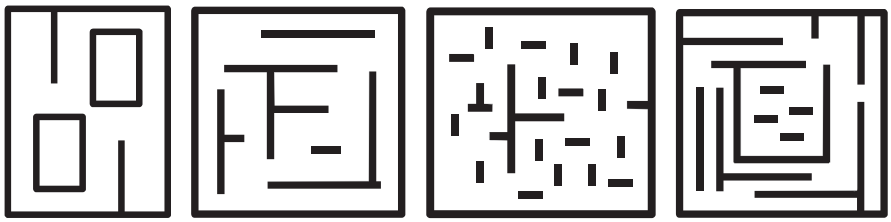

Fig. 5.21 The four arenas used in the second series of experiments; **(a)**: arena 1, **(b)**: arena 2, **(c)**: arena 3, **(d)**: arena 4.

The $(\mu+1)$-ONLINE algorithm allows individual robots to maintain a larger population. We hypothesise that using larger populations has a positive effect on the quality of the evolved controllers and test this hypothesis by experiments using $\mu = 1, 3, 9, 13$ in each of the four arenas. The overview of experimental details is given in Table 5.3.

Table 5.3

Experiment details	
Task	fast forward
Arena	4 different arenas, see Fig 5.21
Robot group size	5
Simulation length	1000 time steps
Controller details	
ANN type	perceptron
Input nodes	9 (8 sensory inputs and 1 bias node)
Output nodes	2 (left and right motor values)
Evolution details	
Representation	real valued vectors with $-4 \leq x_i \leq 4$
Chromosome length L	18
Fitness	See equation 5.1
Recovery time	30 time steps
Evaluation time	30 time steps
$P_{reevaluate}$	0.2
Population size μ	1, 3, 9, 13
Mutation	Gaussian $N(0, \sigma)$ with adaptive σ values, $\sigma_{initial} = 1$
Crossover	none
Parent selection	random
Survivor selection	replace worst when better

Fig. 5.22 shows the development of champion performance as evolution progresses for typical runs with varying values of μ. We clearly see that lower values of μ display considerable drops in champion fitness–much more so than large values. Such drops are, as noted in Sect. 5.2.5.2 the effect of the inherently noisy fitness calculation when an actually quite poorly performing genome is evaluated as having a high fitness due only to auspicious circumstances. Obviously, small populations are more susceptible to removing a good individual after evaluating such a lucky challenger. For example, for $\mu = 10$, evolution would have to encounter 10 lucky challengers before actually removing the champion, but with $\mu = 1$, the champion is dropped immediately.

By the same token, the champion fitness at the end of the runs as shown in Fig. 5.23 is better for larger values of μ: good individuals are more easily forgotten for low values of μ, thus more runs will end with low champion fitness. The peak performance (best champion ever), however, does not vary with μ.

To analyse actual robot performance, we show the average fitness including challengers and re-evaluations over the last 20 evaluations in Fig. 5.24. Here, we see that the actual performance does not increase with μ–in fact, it is worse. This may be explained by the fact that, just as it takes time to forget a good champion, it takes time to forget lucky but actually bad genomes that made it into the population. We expect to be able to mitigate this effect by introducing non-random parent selection, reducing the likelihood of selecting poorly performing genomes from the popula-

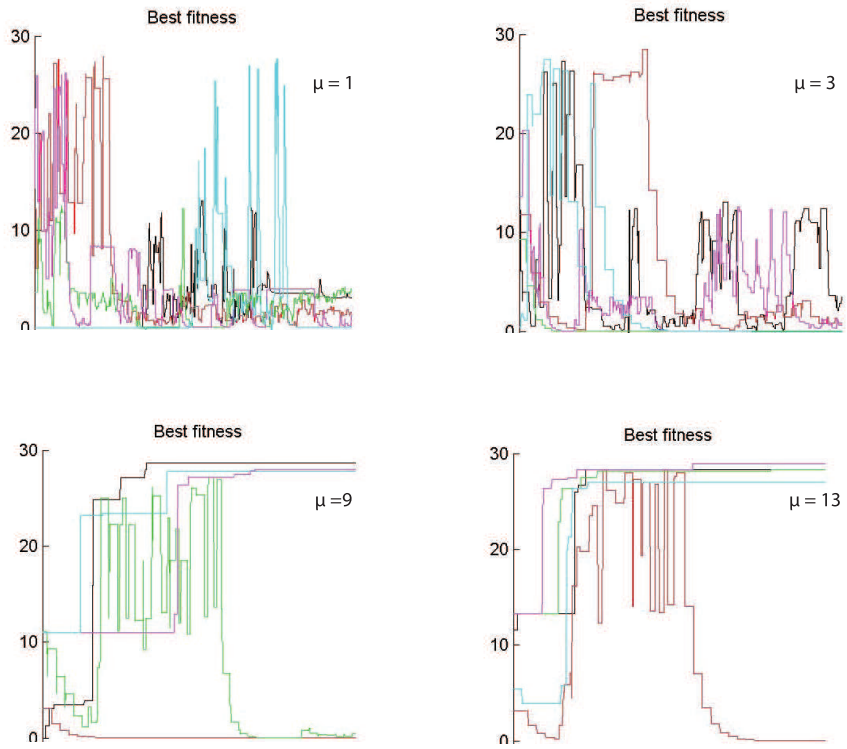

Fig. 5.22 Typical runs for $\mu = 1, 3, 9, 13$. The y-axis shows the performance of the current champion, the y-axis the number of generations . Each graph shows five plots; one for each of the five robots in the arena.

tion. Also, increasing the re-evaluation rate (fixed at 0.02 for these experiments) is likely to reduce the time needed to recognise poorly performing genomes.

5.2.6 Conclusions and Future Work

In this section we presented a new taxonomy to classify evolutionary robotics applications, based on three main features belonging to the "when", "where", and "how" dimensions. We deliberately focused on systems with *in vivo* (embodied) fitness evaluations, where the evolutionary algorithm is running on-line without human intervention, and the evolutionary operators are managed on-board either in a centralised/encapsulated or distributed or mixed fashion.

We also reported on the first experiments with on-line, on-board, encapsulated evolution using different population sizes (not to be confused with the number of robots) by means of a feasibility study within the projects. The results indicate that

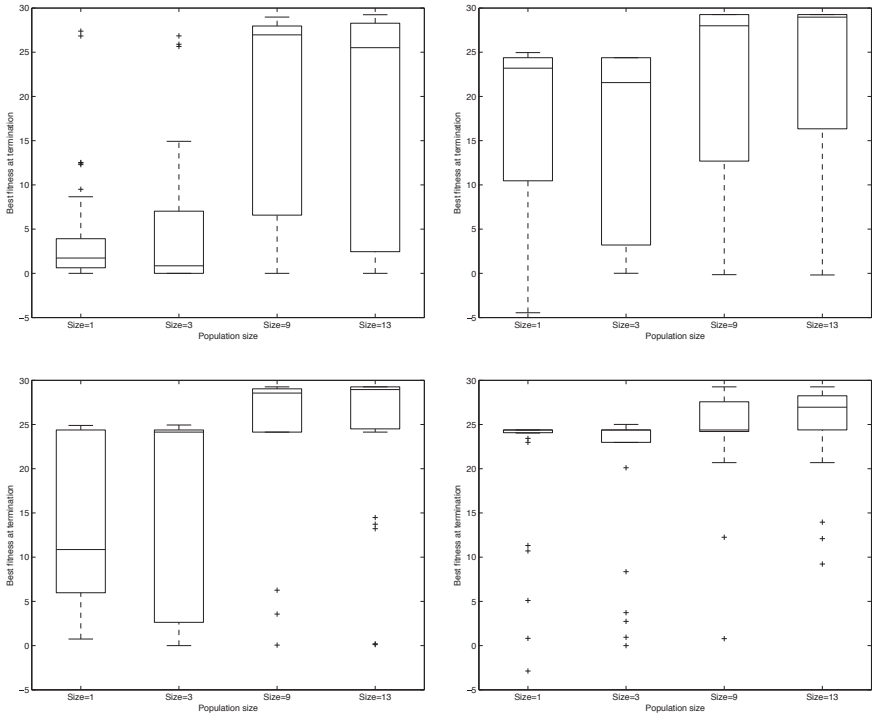

Fig. 5.23 The effect of increasing the population size μ on champion fitness. Each box summarises champion performance in 10 runs with μ set to 1, 3, 9 or 13; the central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. Top-left: arena 1, top-right-arena 2, lower-left: arena 3, lower-right: arena 4.

the approach is feasible even in a most simple setup, with random parent selection and no crossover, and show that increasing the population size improves the quality of evolved controllers. Currently we are conducting experiments with advanced evolution strategies with covariance matrix adaptation that form a promising option if the controllers can be represented by real-valued vectors. In the future we will investigate the evolution of controllers using distributed evolution and the combination of the encapsulated and distributed variants (island model with migration).

In conclusion, we can state that the two greatest challenges embodied, on-line, on-board evolutionary systems have to face is the short time period for evolution and the very noisy fitness evaluations. Technically speaking, the robots can only evaluate a few candidate solutions (genomes) in total and cannot perform enough re-evaluations. The source for both problems lies in the *physical* constraints of the system: a robot can only use one controller at a time and it should be using it for a "long" period to gain solid information about its quality. Thus, a possible cure for these two challenges is circumventing those physical constraints by incorporating

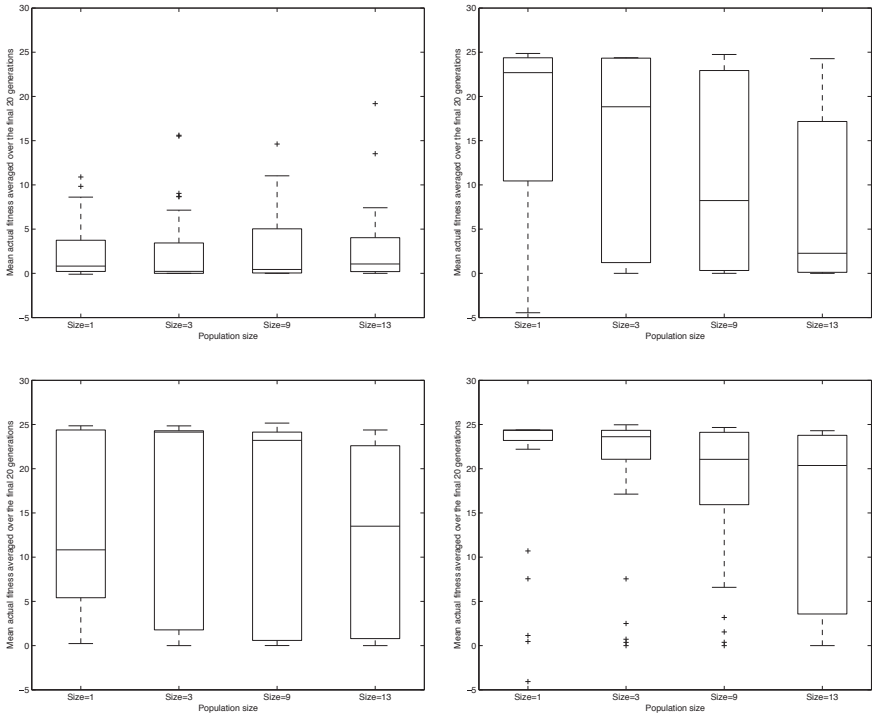

Fig. 5.24 The effect of increasing the population size μ on actual fitness. Each box summarises actual performance in 10 runs with μ set to 1, 3, 9 or 13; the central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually. Top-left: arena 1, top-right-arena 2, lower-left: arena 3, lower-right: arena 4.

and using a (possibly rough) simulator inside the robots for preliminary candidate assessment; only genomes that pass this quick test are further evaluated in real life. The costs will occur in the increased storage, memory and CPU power. The benefits will be the increased number of candidate solutions that can be evaluated and the increased number re-evaluations per candidate solution. It is very likely that both will contribute to more powerful evolutionary search still fitting within the limited physical time frame.

References

[Schut et al., 2009] Schut, M., Haasdijk, E., & Eiben, A.E. 2009. What is Situated Evolution? In: Tyrrell, A. (ed), Proc. of the IEEE Congress on Evolutionary Computation (CEC 2009). Trondheim, Norway: IEEE Press.

[Watson et al., 2002] Watson, Richard A., Ficici, Sevan G., & Pollack, Jordan B. 2002. Embodied Evolution: Distributing an evolutionary algorithm in a population of robots. Autonomous Systems, 39(1), 1–18.

[Nehmzow, 2002] Nehmzow, U. 2002. Physically Embedded Genetic Algorithm Learning in Multi-Robot Scenarios: The PEGA Algorithm. In: Prince, C.G., Demiris, Y., Marom, Y., Kozima, H., & Balkenius, C. (eds), Proc. of The Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems. Lund University Cognitive Studies, no. 94. Edinburgh, UK.

[Usui & Arita, 2003] Usui, Y., & Arita, T. 2003. Situated and Embodied Evolution in Collective Evolutionary Robotics. Pages 212–215 of: Proc. of the 8th International Symposium on Artificial Life and Robotics.

[Schwefel, 1995] Schwefel, H.-P. 1995. Evolution and Optimum Seeking. Wiley, New York.

[Beyer, 2000] Beyer, H.-G. 2000. Evolutionary algorithms in noisy environments: theoretical issues and guidelines for practice. Computer Methods in Applied Mechanics and Engineering, 186(2–4), 239–267.

[Bredeche et al., 2009] Bredeche, N., Haasdijk, E., & Eiben, A.E. 2009. On-line, On-board Evolution of Robot Controllers. In: Proc. of the 9th international conference on Artificial Evolution (Evolution Artificielle 2009). Lecture Notes in Computer Science. Springer-Verlag.

[Nolfi & Floreano, 2000a] Nolfi, S., & Floreano, D. 2000a. Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. Cambridge, MA. / London: The MIT Press.